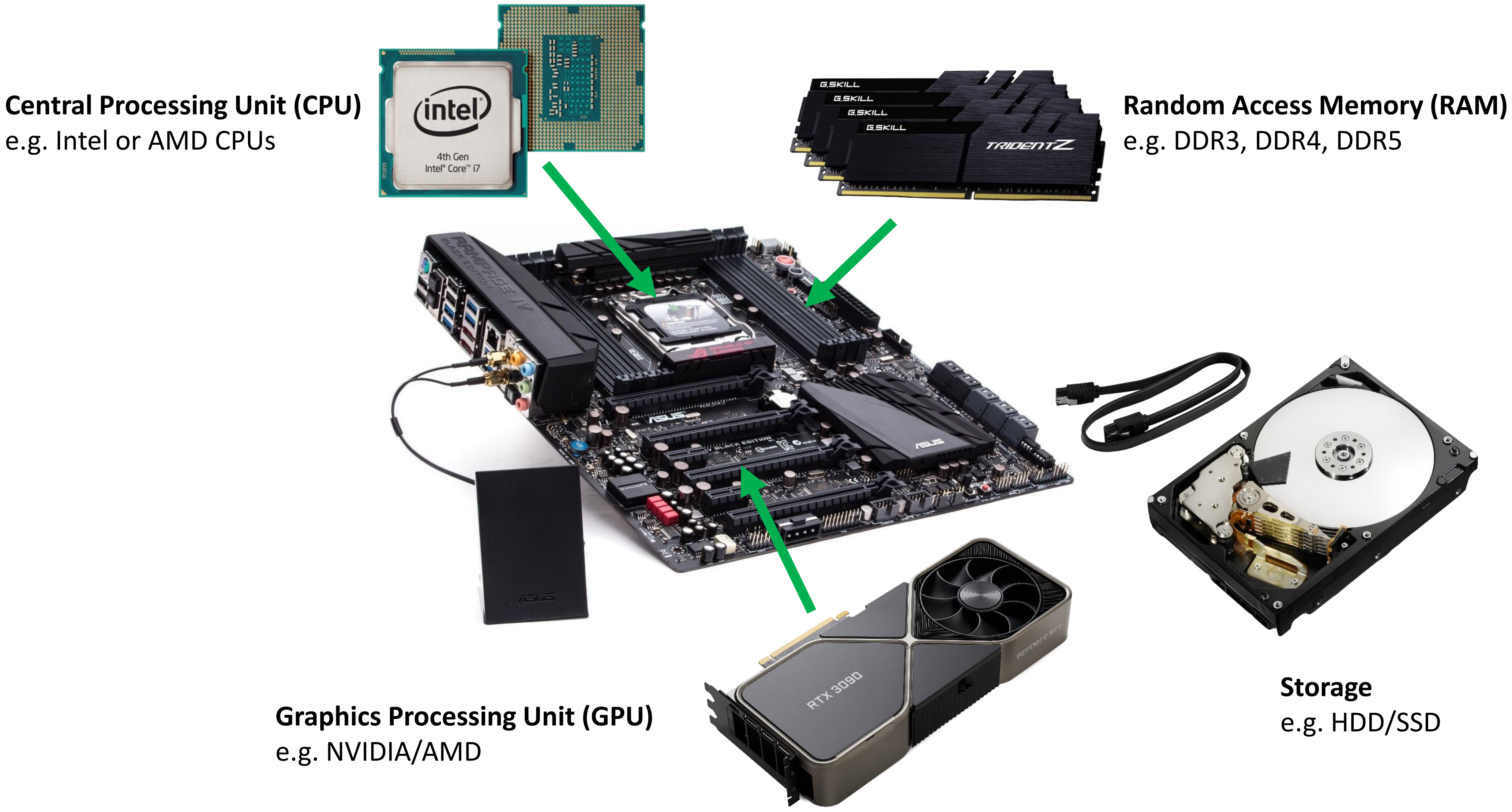
High Performance Computing in Julia
from the ground up.

Introduction to GPU Programming & CUDA.jl

9/10

Aims

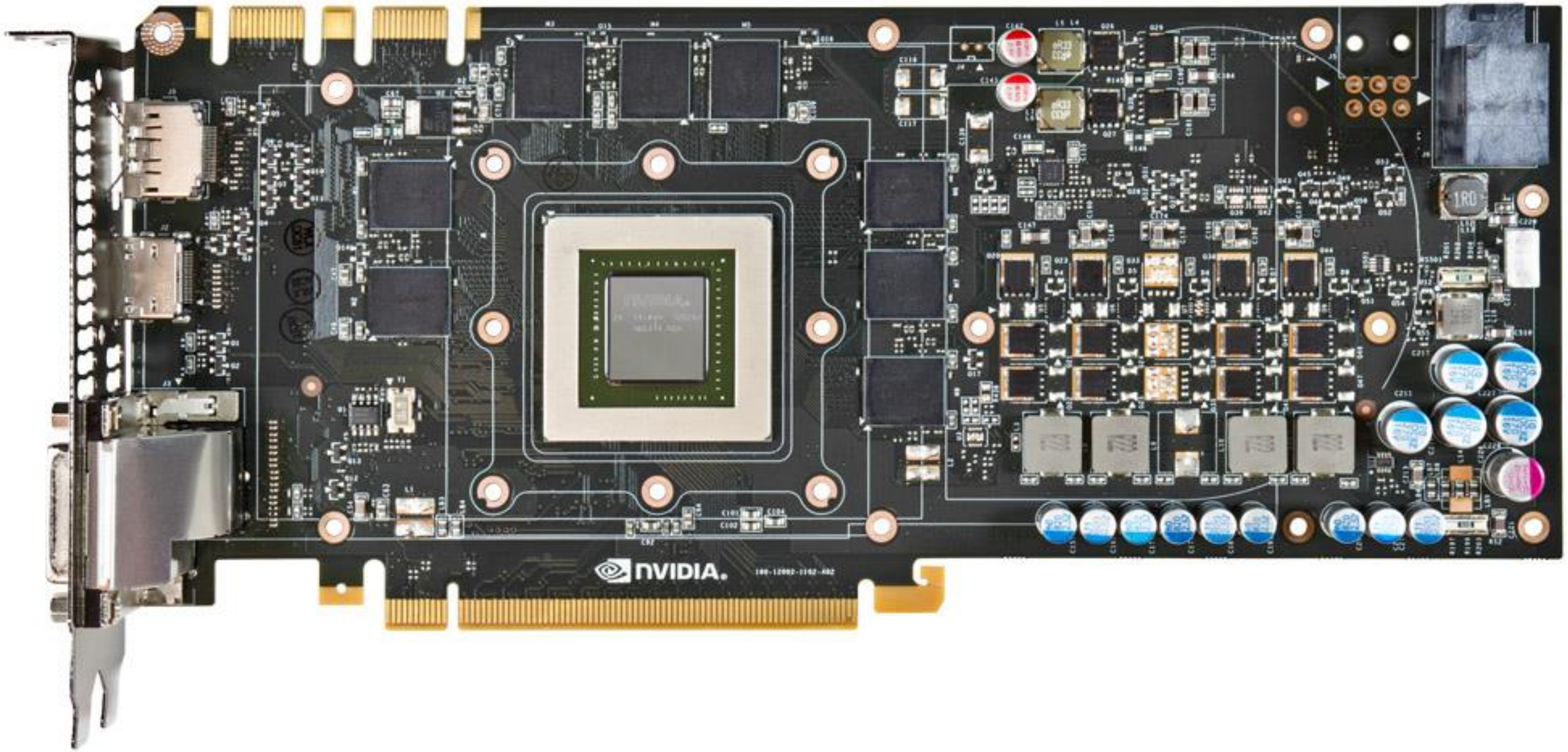
- To introduce GPU hardware
- To discuss the different options for GPU programming
- To introduce array-based GPU programming with CUDA


Central Processing Unit (CPU)
e.g. Intel or AMD CPUs
intel
4th Gen
Intel® Core™ i7
Random Access Memory (RAM)
e.g. DDR3, DDR4, DDR5
G.SKILL
TRIDENTZ
ASUS
RTX 3090
Graphics Processing Unit (GPU)
e.g. NVIDIA/AMD
Storage
e.g. HDD/SSD

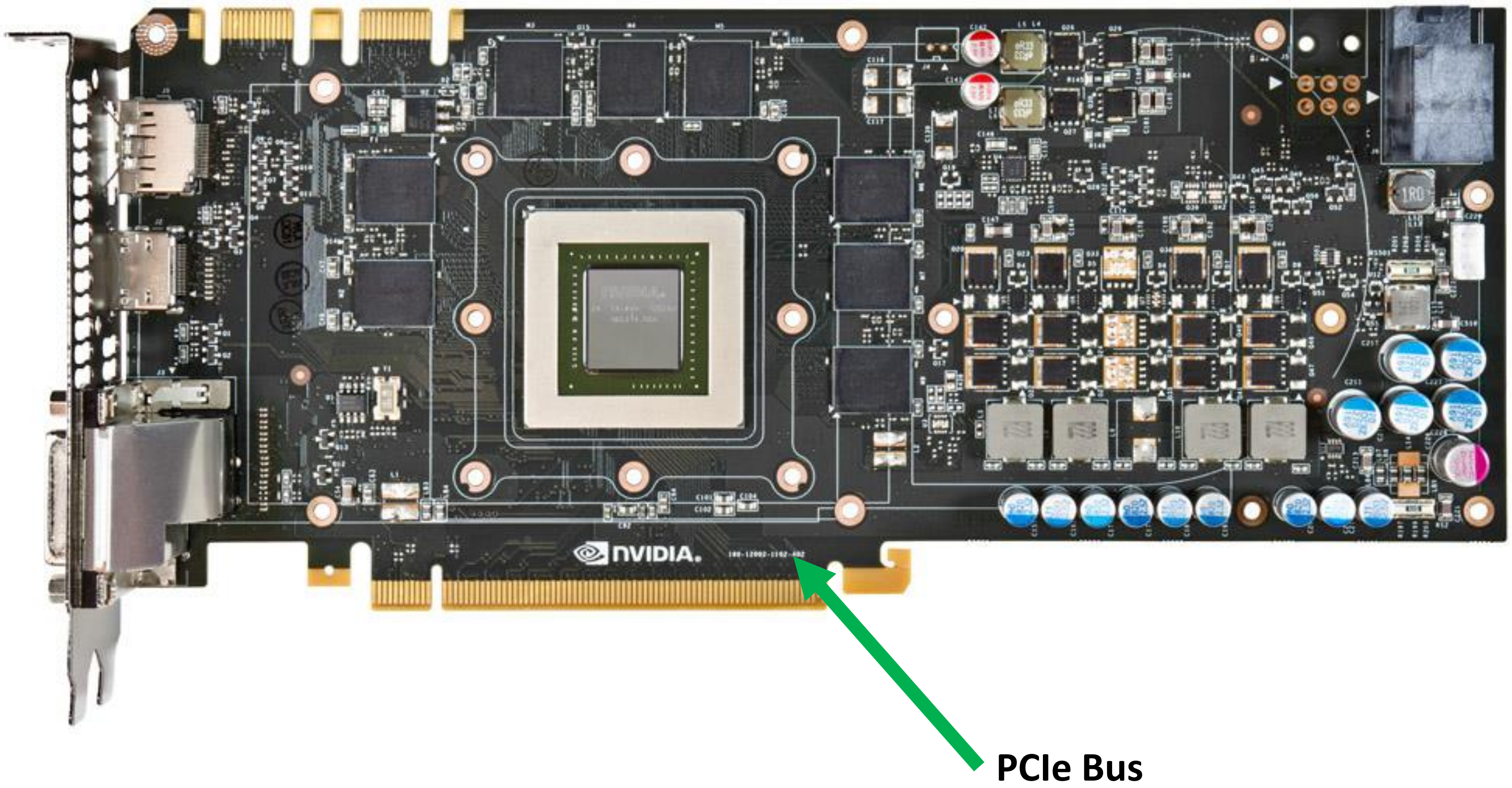
General-Purpose Graphics Processing Unit (GPGPU)

- Massively parallel co-processors which work alongside the CPU
- Traditionally used to accelerate graphics workloads
- Hardware is also useful for accelerating many modern workloads (e.g. AI/ML/Fluid dynamics etc)
- While CPUs has 10s of cores, GPUs have 10,000s of cores

Anatomy of a GPU

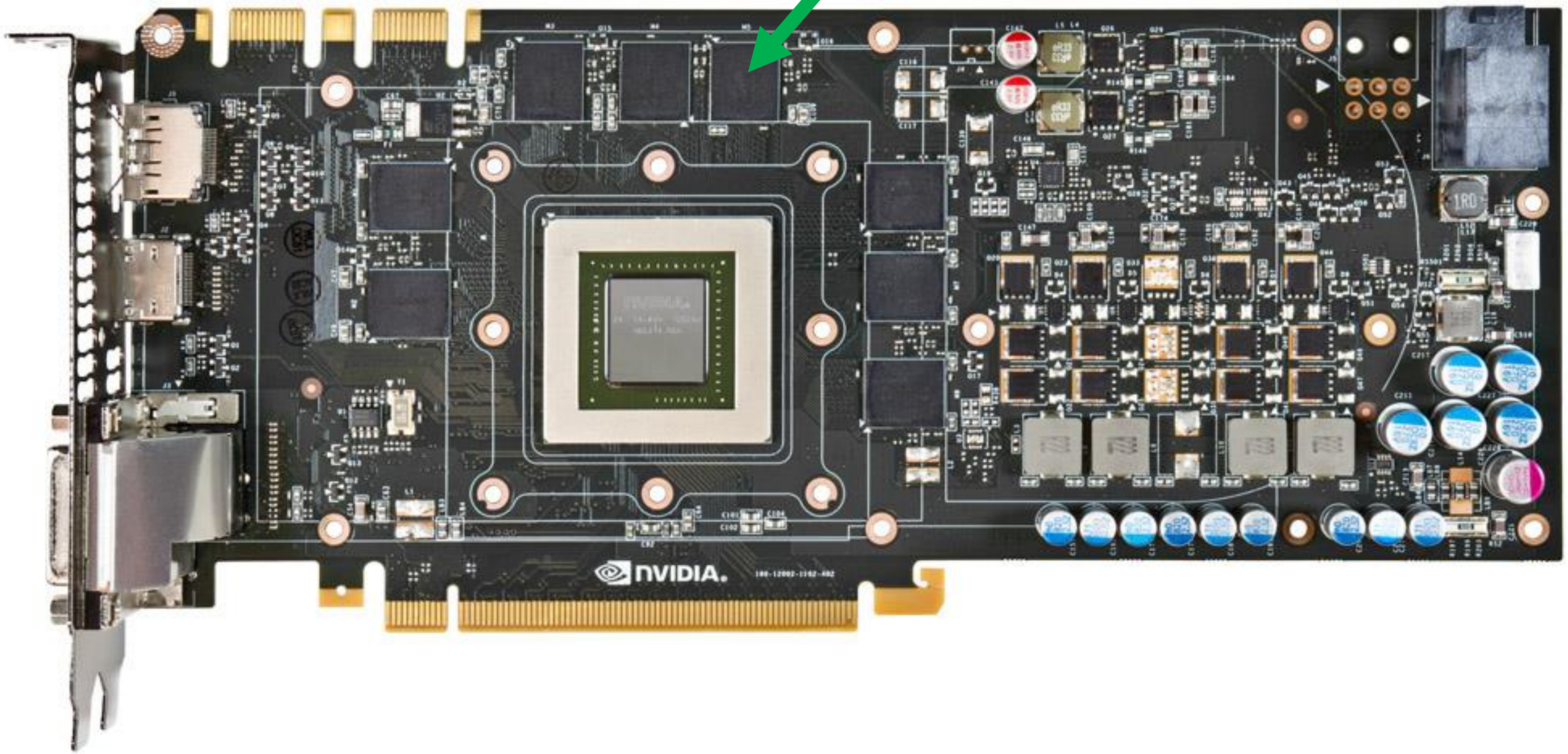


Anatomy of a GPU



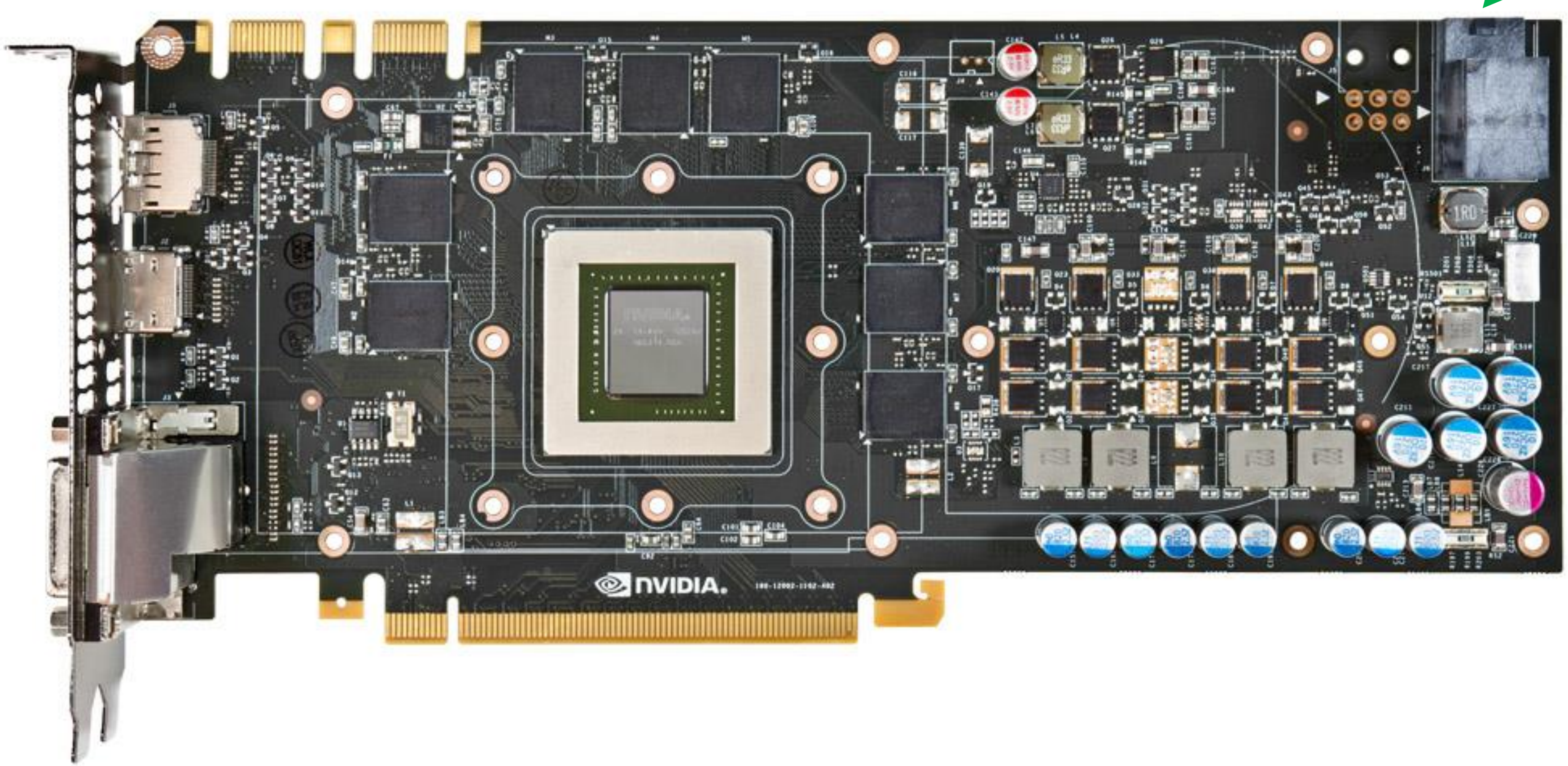
Anatomy of a GPU

DRAM
e.g. GDDR6/HBM2

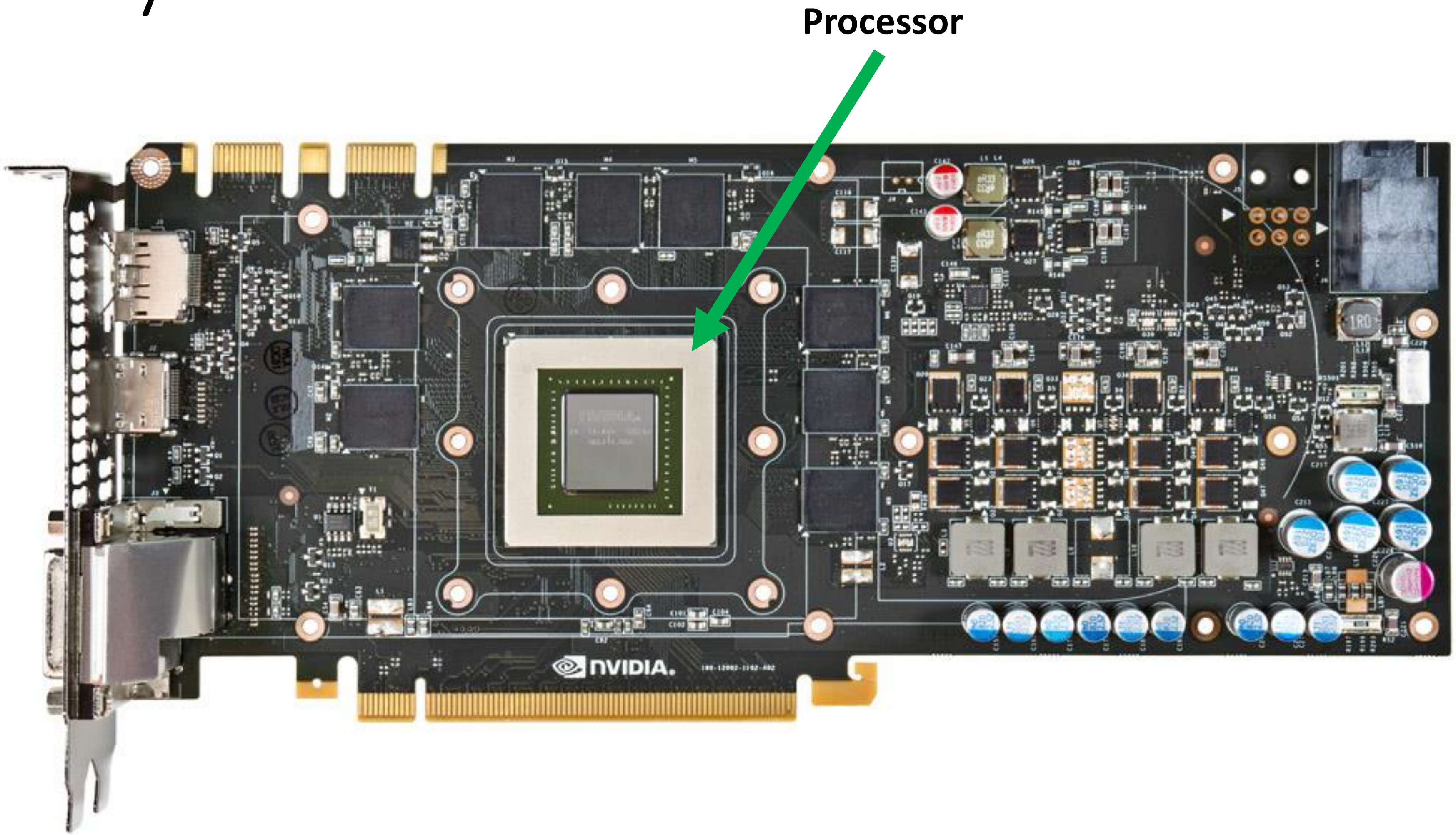


Anatomy of a GPU

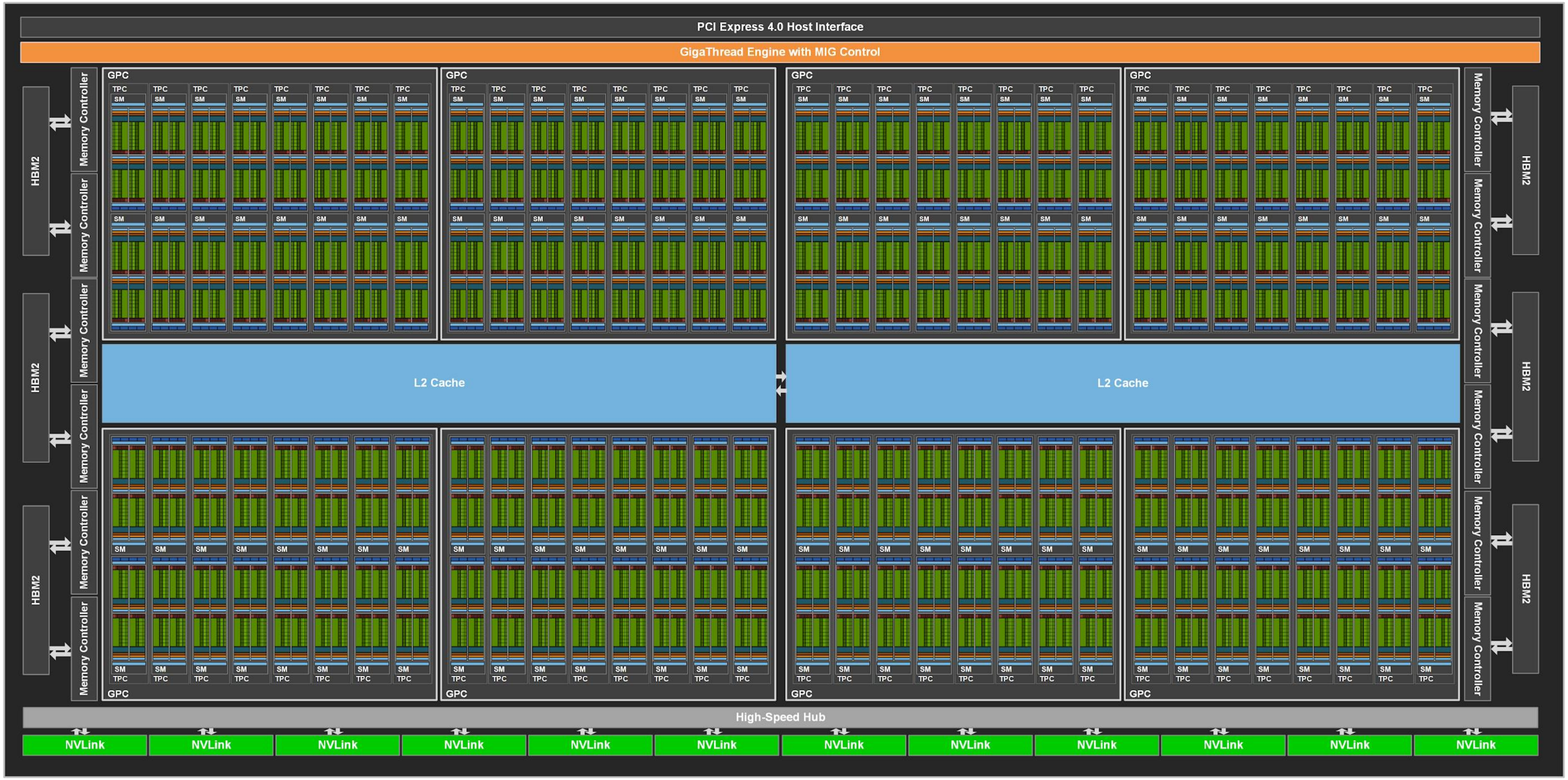
Power Delivery



Anatomy of a GPU



Anatomy of a GPU: **A100**



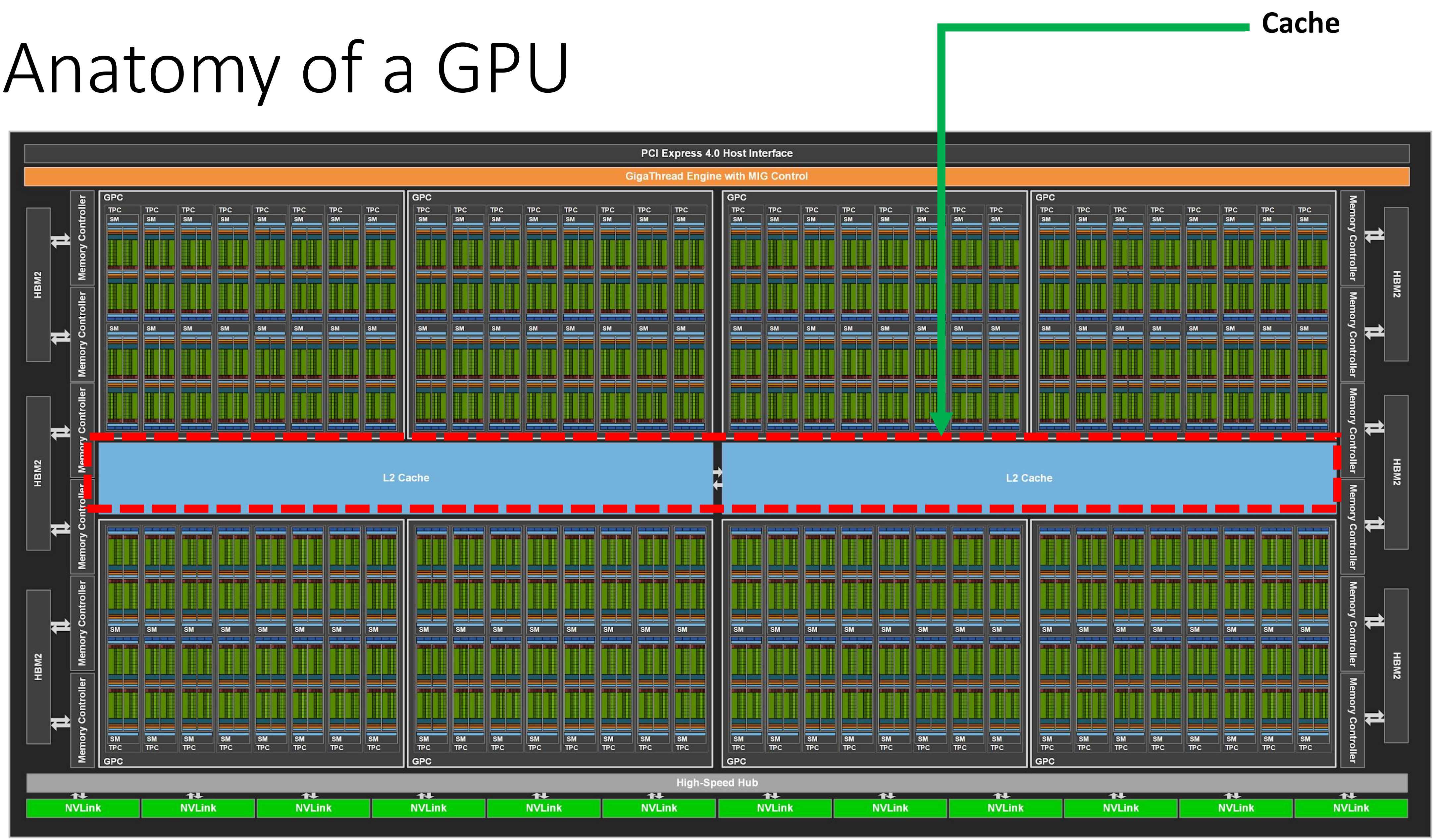
https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/

Anatomy of a GPU: **A100**

HBM2
Memory

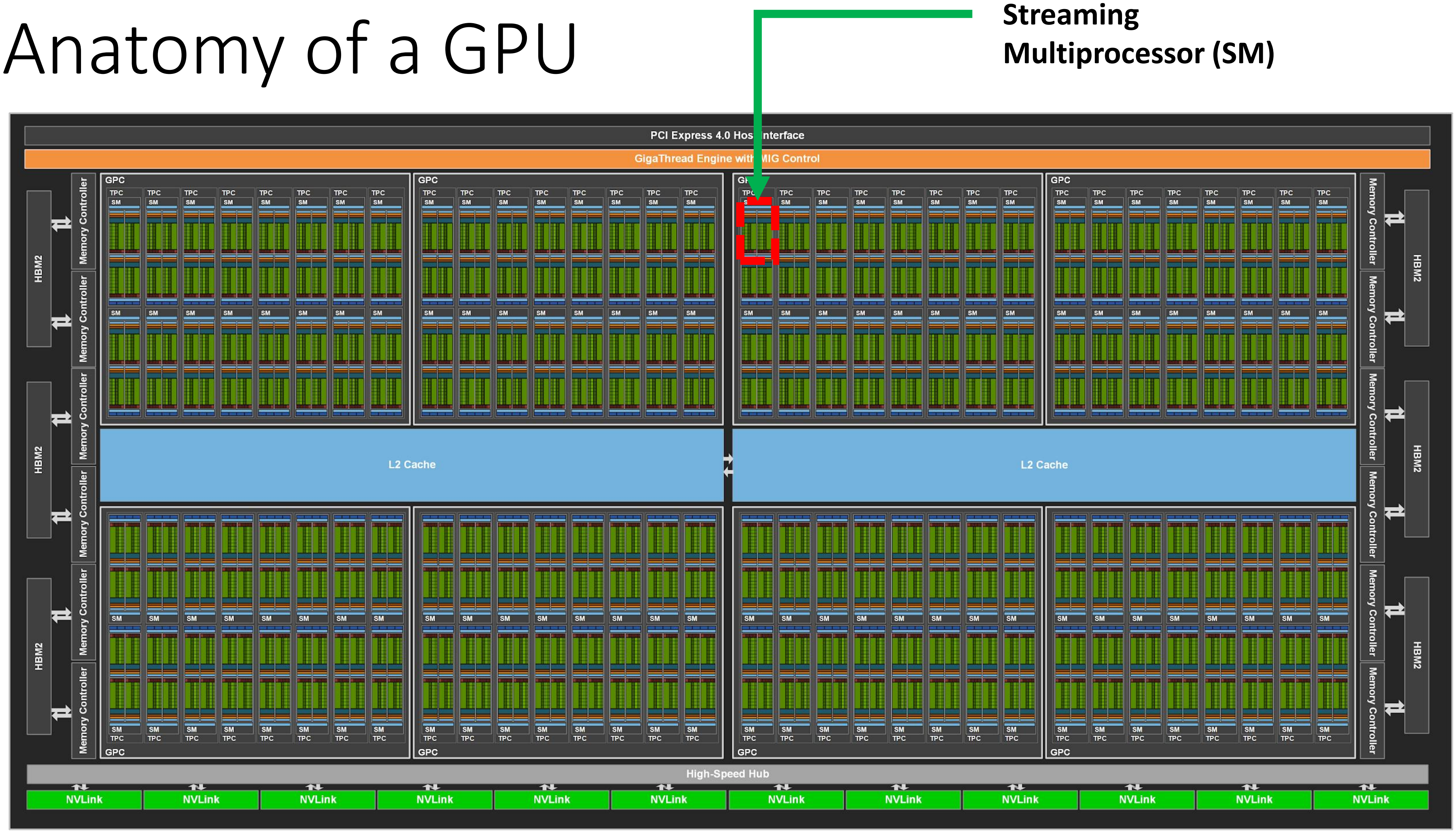
https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/

Anatomy of a GPU



https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/

Anatomy of a GPU



https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/

GPU Cores

- Contains many “cores” which can be used to perform math operations
- Less general purpose and slower than a CPU “core”
- Focuses on floating-point and integer math (32 bits)
- Has a much smaller cache than a CPU core

https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/

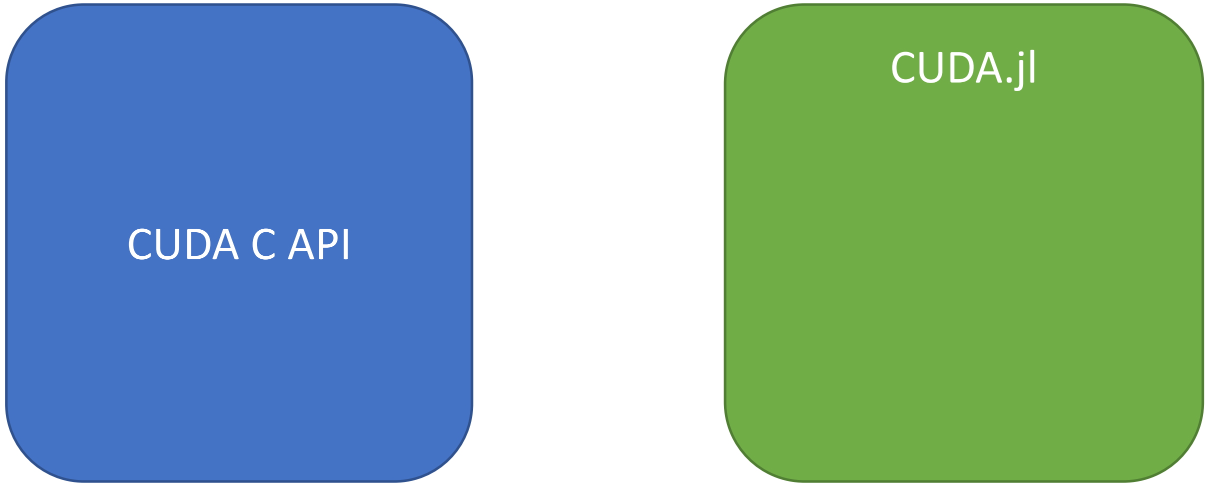
GPU Platforms

CUDA

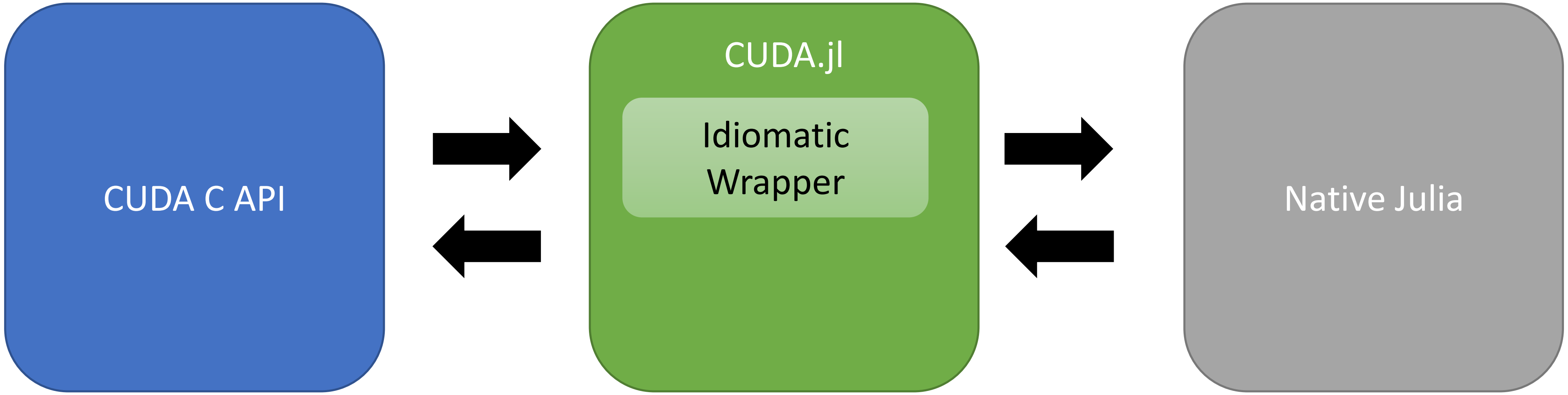
- NVIDIA has the most market share in HPC GPU applications (especially in AI/ML)
- CUDA is the most popular **hardware platform** and **API** for programming GPUs
- Provides an **abstraction** to allow programming for many different GPU models, providing a virtual instruction set to the programmer
- Has GPU implementations of common algorithms such including **BLAS**, **FFT**, **RNG** and **linear solvers**
- More recently, has **CUDNN** for machine learning acceleration

CUDA.jl

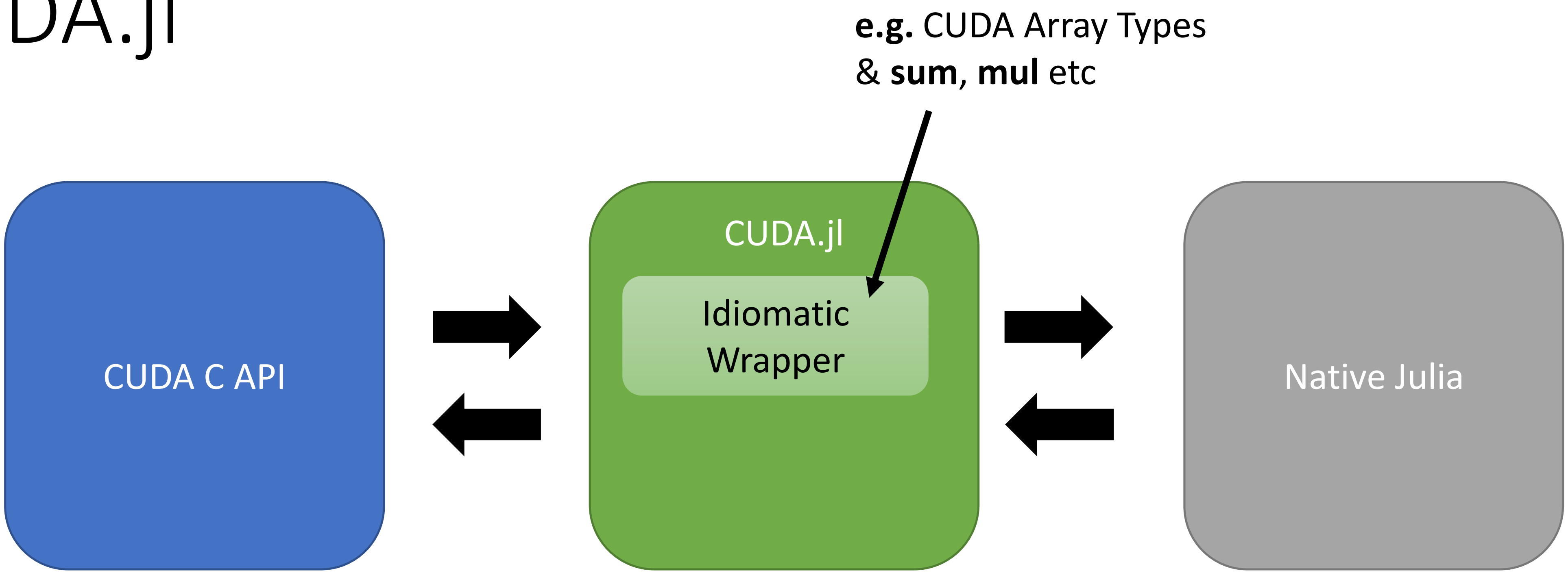
CUDA.jl



CUDA.jl

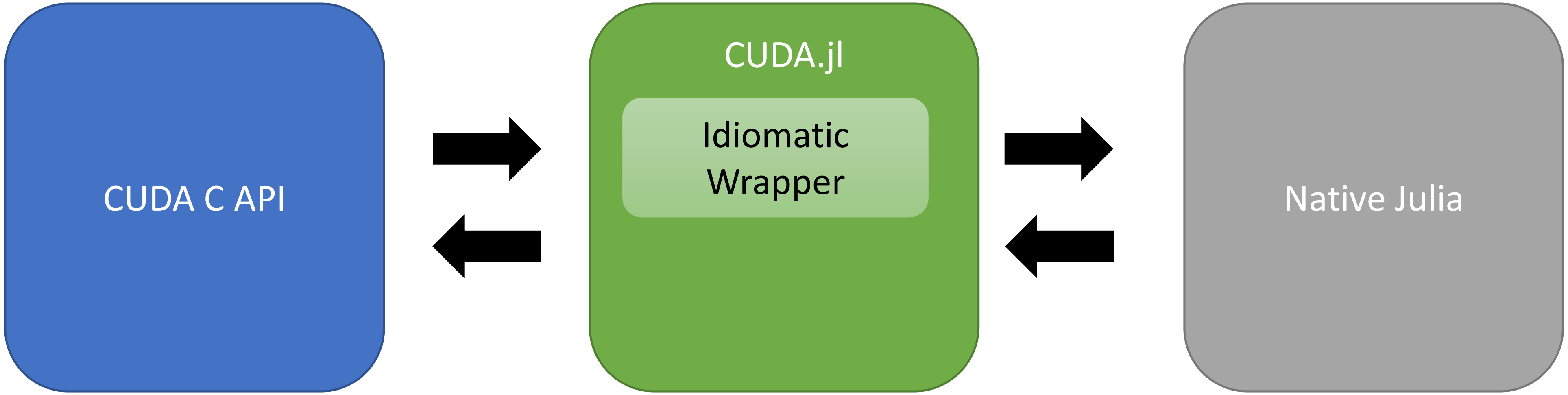


CUDA.jl

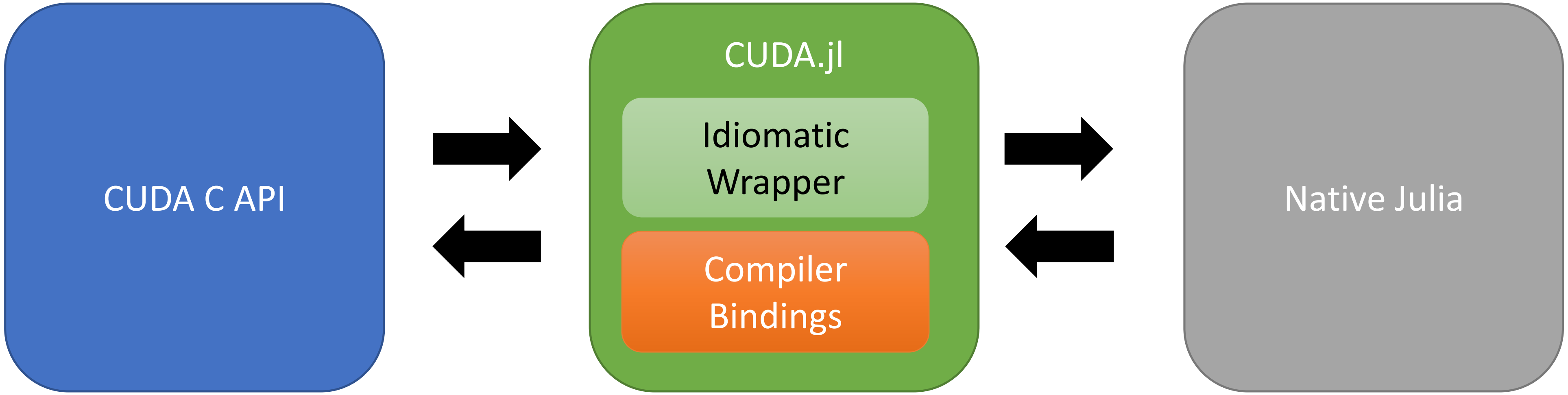


e.g. CUDA Array Types
& **sum**, **mul** etc
CUDA C API
CUDA.jl
Idiomatic Wrapper
Native Julia

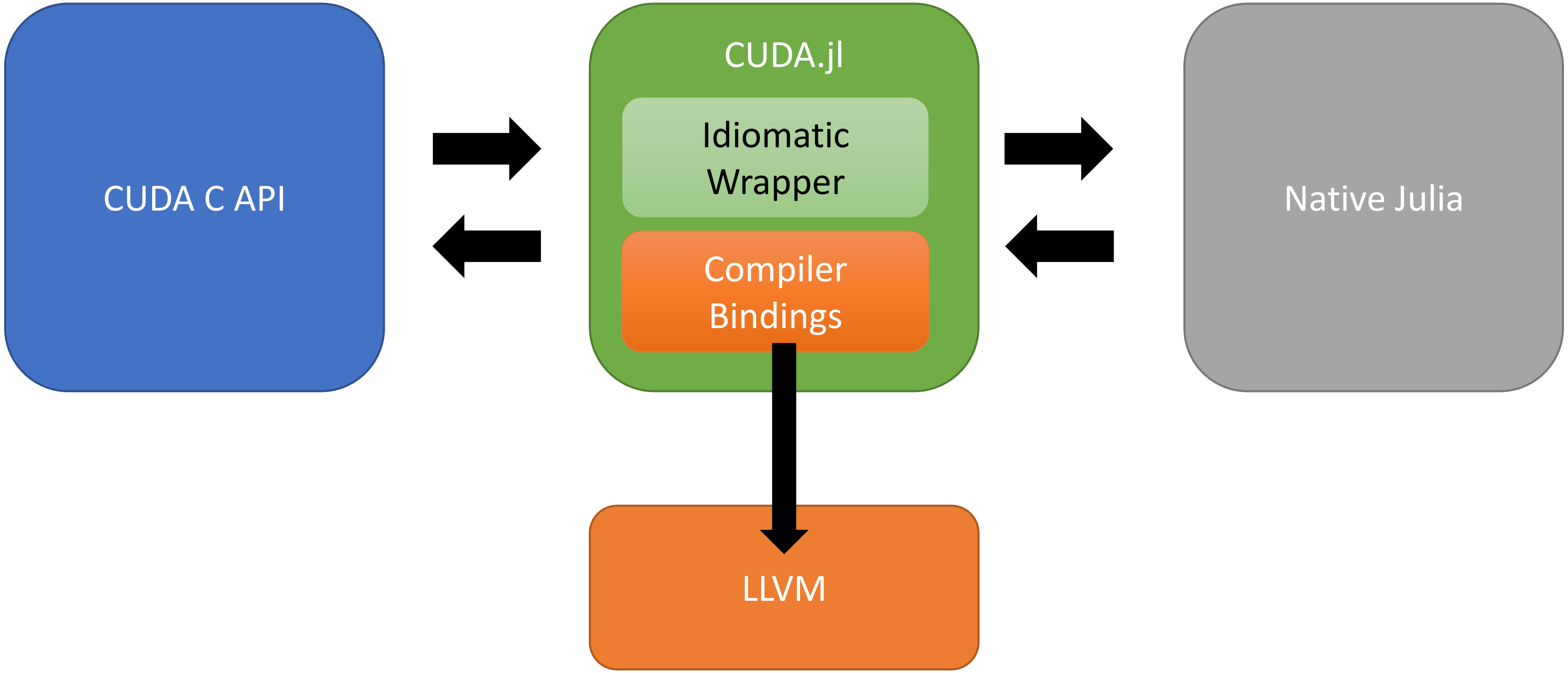
CUDA.jl



CUDA.jl

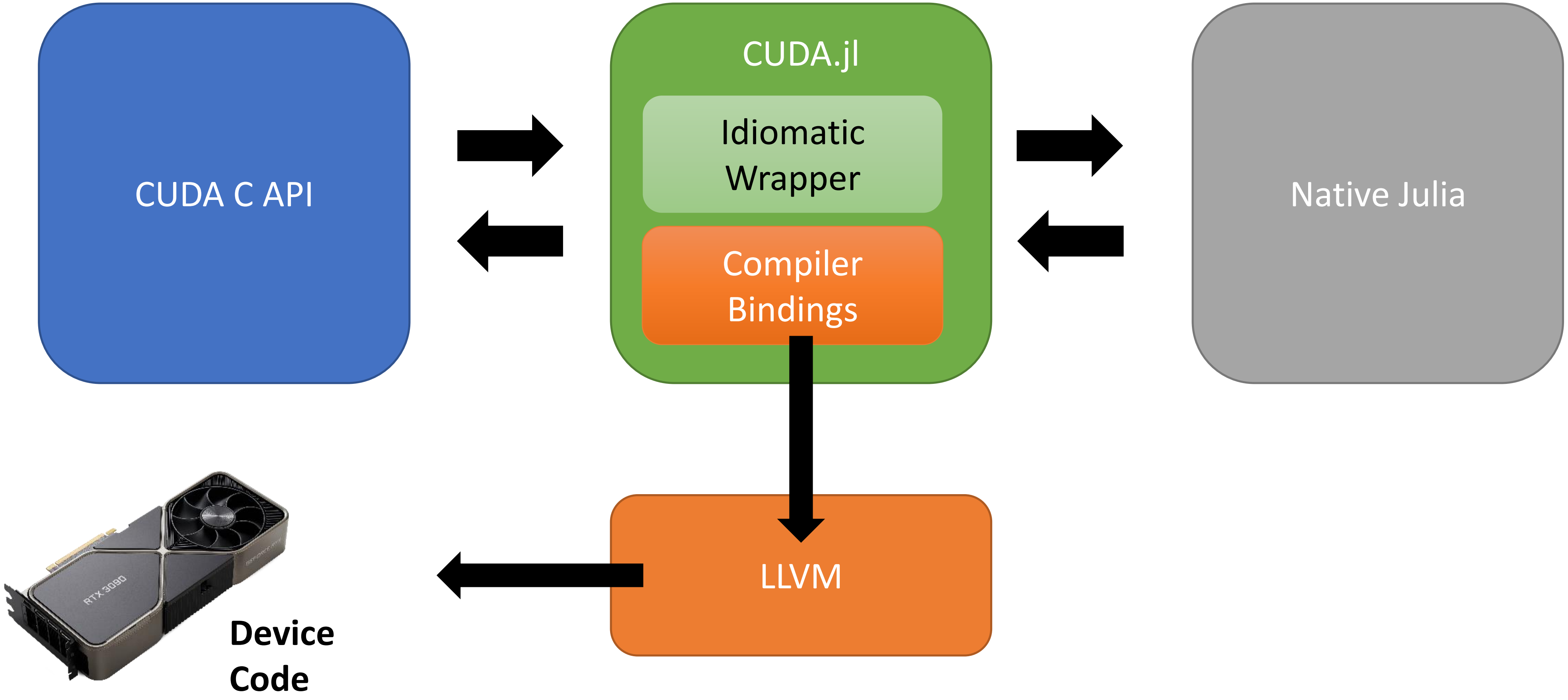

CUDA C API
CUDA.jl
Idiomatic Wrapper
Compiler Bindings
Native Julia

CUDA.jl



CUDA C API
CUDA.jl
Idiomatic Wrapper
Compiler Bindings
Native Julia
LLVM

CUDA.jl

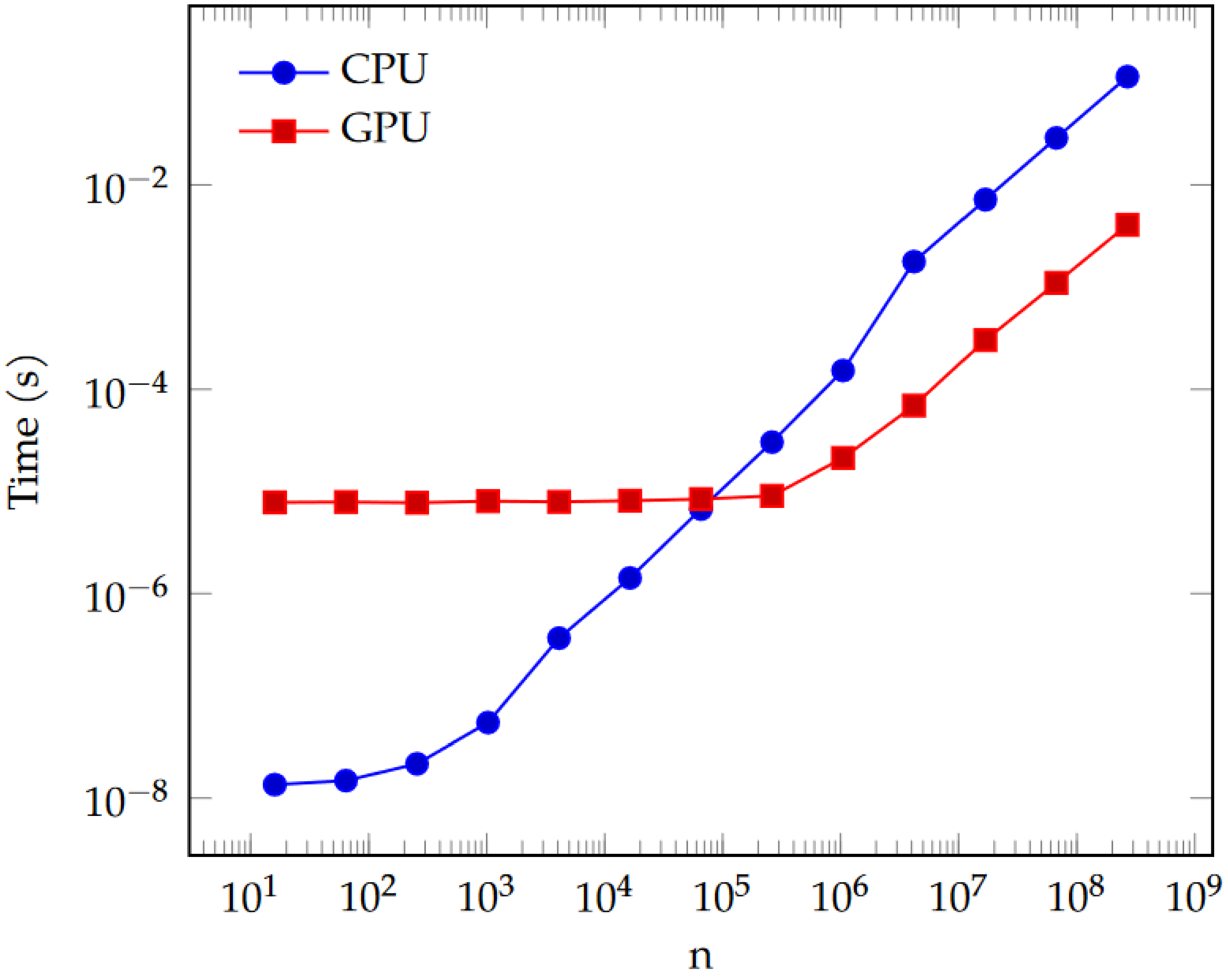

CUDA C API
CUDA.jl
Idiomatic Wrapper
Compiler Bindings
Native Julia
LLVM
RTX 3090
Device Code

CUDA.jl

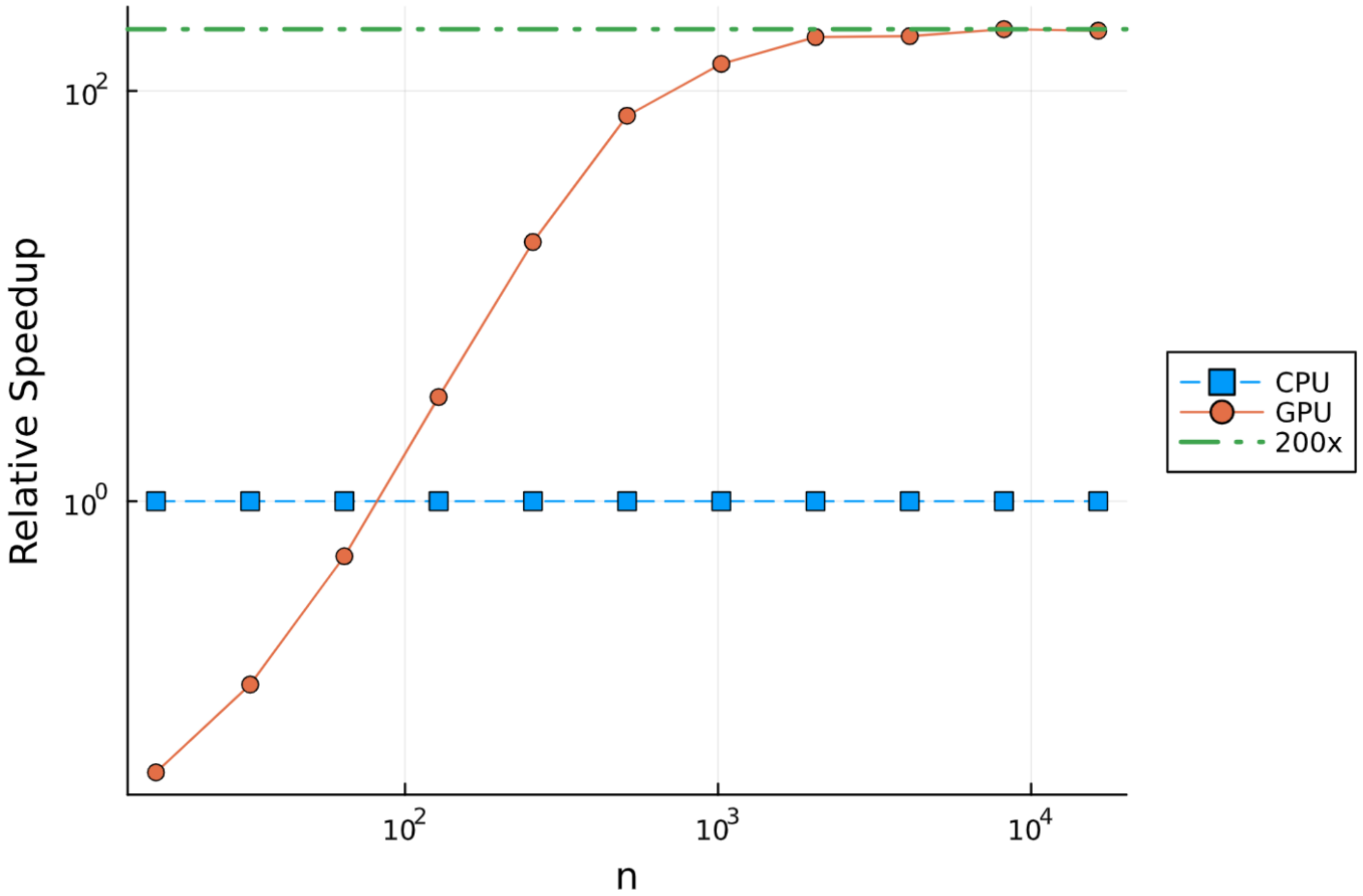
Array Programming

https://github.com/MPAGS-HPC-in-Julia/gpu-demo

Vector Addition: $\mathcal{O}(n)$



Matrix Multiplication: $\mathcal{O}(n^3)$



Array Programming with CUDA.jl

Restriction

- Scalar indexing on GPU transfers data between GPU and CPU and is **very slow**
- All memory required by the computation on the GPU **must** already exist on the GPU
- Broadcasted functions must be **type stable** to be successfully compiled into native GPU code

Tip

- Avoid "**for**" loops and use broadcasting and generic functions
- Use in-place operations where possible to avoid excessive allocation & copying
- Use the "**@code_warntype**" macro to make sure the function is type safe

Generic GPU Programming

- Stick to broadcasting and functions with GPU specialisations such as “**map**”, “**reduce**”, “**sum**” etc
- This allows code to work with both CPU and GPU arrays
- If code is more complex, and requires a “**for**” loop, you must write a custom GPU program – called a **kernel**
- **CUDA.jl** allows us to write kernels in pure Julia, which can get compiled to native device code
- Packages like **KernelAbstractions.jl** can help to write generic device-independent code

Next Session – CUDA.jl Kernel Programming

Assignment

https://classroom.github.com/a/q9ycWkI6

Task:

- Calculate the visualisation for the Julia set fractal using the GPU

Julia Set